<u>**Write-up for the Robo-ND deep learning project**</u>

In this project I implemented a deep-learning network for semantic segmentation at pixel level for identifying a target person so a quadrotor can follow it in the simulator.

My network was an FCN network with three encoder blocks with depth 32, 64, 128. After the encoders I used a 1x1 convolution layer with 256 depth and then three decoders which each one of them does upsampling by a number of two. A final convolution layer was used for classification. I trained the model with the data provided by Udacity and I got an accuracy of 45%. Then I tested the model on the simulator and the quadrotor started following the target as expected.

The FCN architecture is commonly used for image classification and especially for identifying every pixel in the image. The encoder layers are used for identifying useful information from the input image. The first encoder layer is trained to identify simple features and the next encoder layers identify more and more complicated features and shapes.

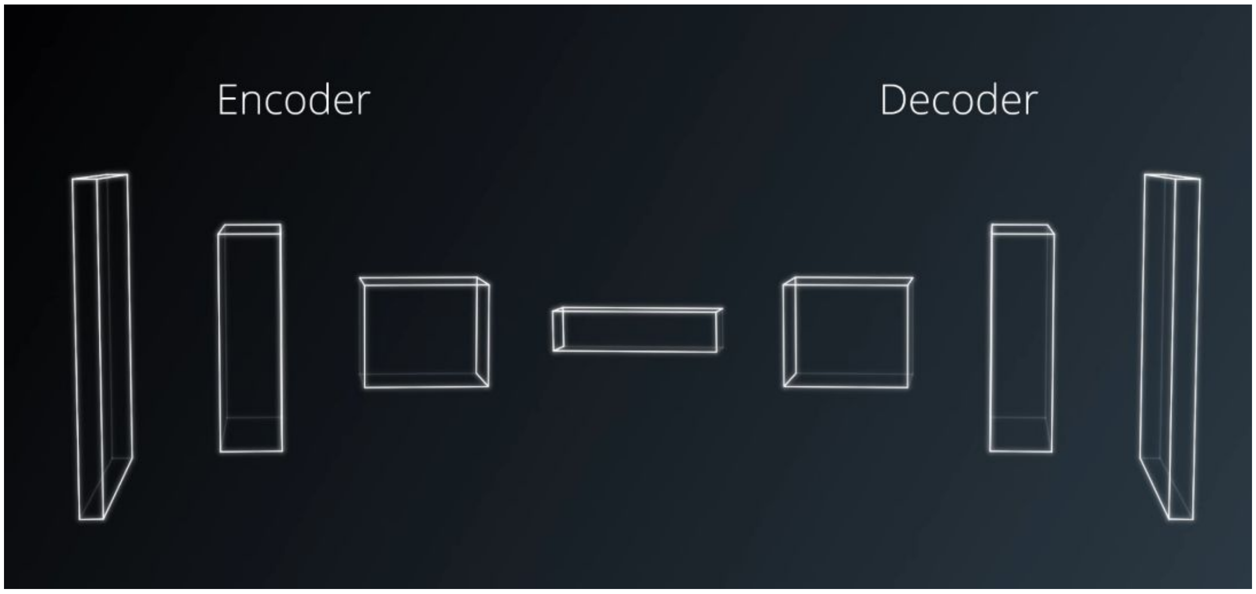
The 1x1 convolutional layer is used when we want to flatten a convolutional layer without loss of spatial information.

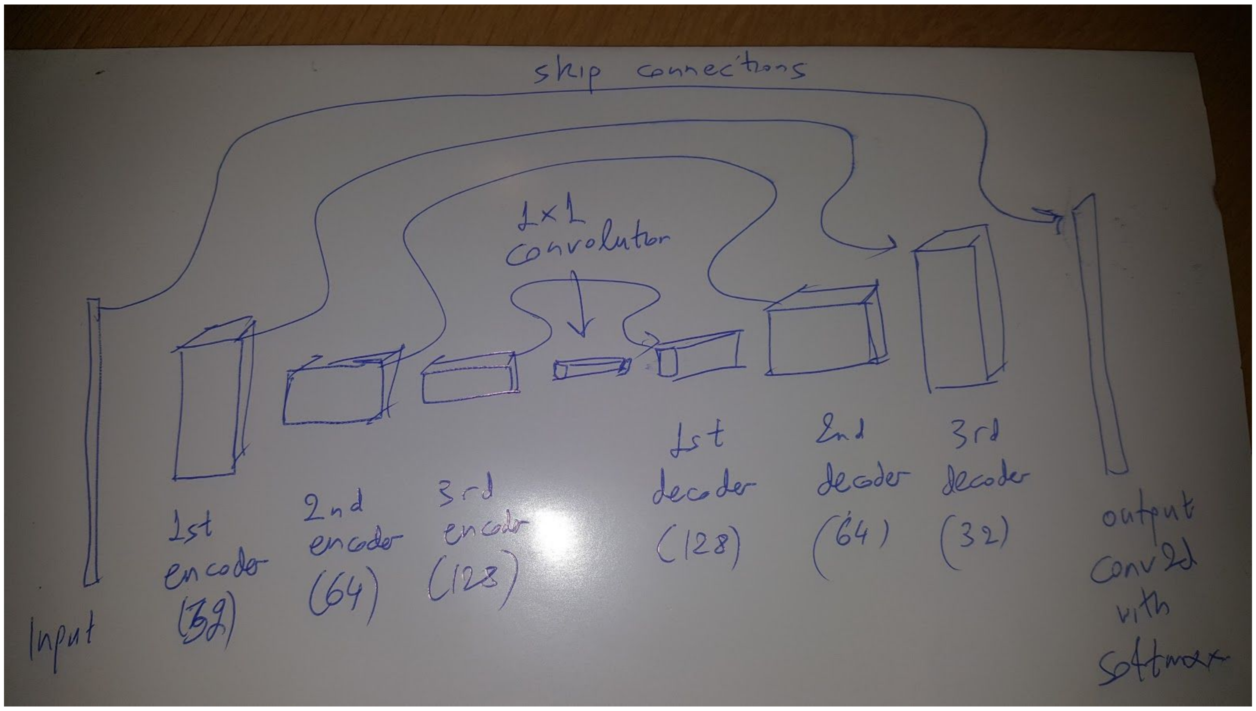
The decoder layers are used to upsample the encoder output back to the original image size.

Also there are 4 skip connections from input to output layer, 1st encoder to 3rd decoder, 2nd encoder to 2nd decoder and 3rd encoder to 1st decoder. These skip connections are used to retain the information that would be lost from the encoding to decoding process.

The last convolutional layer of the model is used to make the classification of each pixel using the softmax activation function.

The model architecture





Each encoder block consists of a separable convolution with batch normalization. This was implemented in the encoder_block function using:

```
output_layer = separable_conv2d_batchnorm(input_layer, filters, strides)
```

The decoder block was implemented in the decoder_block function and consisted of :

Upsample the small input layer using bilinear_upsample() and the concatenate it with the large input layer.

After that I added three separable convolution layers with batch normalization. I found that 2 to 3 convolution layer work well on training the network.

The FCN model was created inside the fcn_model function.

It consists of 3 encoder blocks that each block increase the depth , then a 1x1 convolution layer with batch normalization followed by three decoder blocks.

The final layer is a 2D convolution layer with softmax activation function and ‘same’ padding.

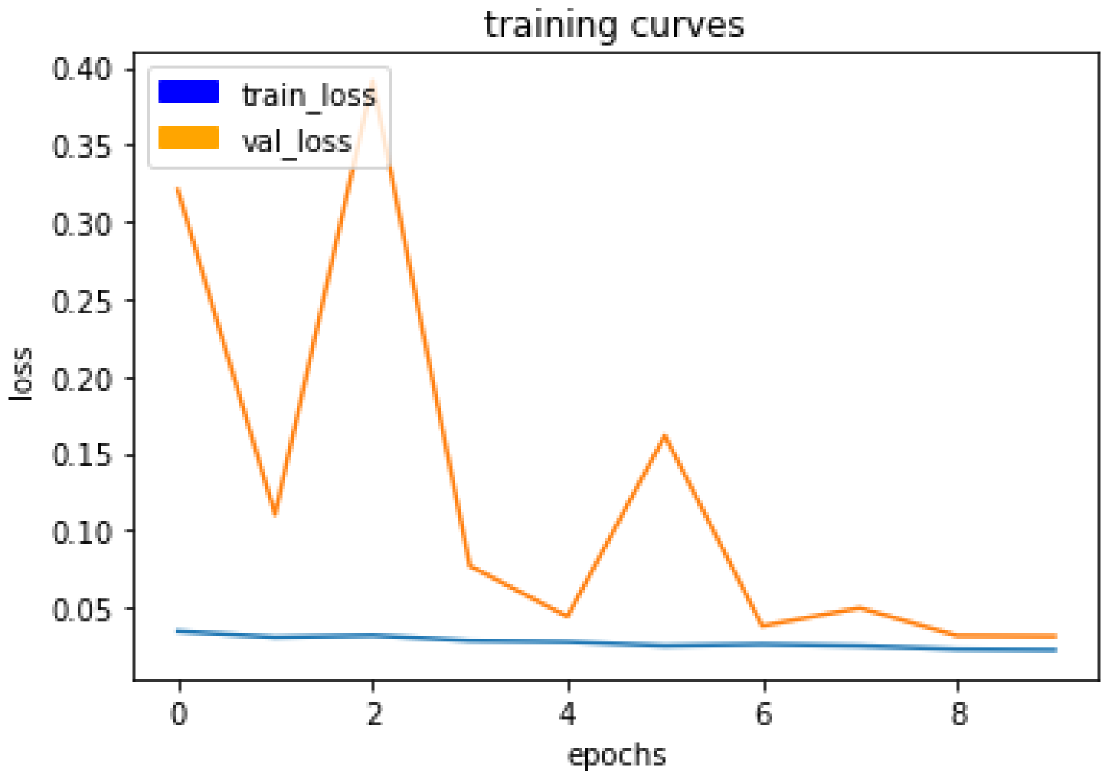
I tried different FCN models for training but I observe that using more than three encoder blocks the model was overfitting. With fewer blocks I got an accuracy of 32%.

While training the network I used for 5 to 20 epochs using my GPU GTX960M. I didn’t use more than 20 epochs because the training time was too big and I didn’t get any additional benefit.

I tried learning rate of 0.1, 0.01 and 0.001 and I got the best results with 0.01.
Batch size was set to 32 and num_epochs = 10.
I left steps_per_epoch = 100 and validation_steps = 50 constant throughout my different training tries for computation issues.

For workers > 2 my GPU was crushing so I kept it at two.

The loss graph of my final model that achieved 45% accuracy is shown below. I was afraid that it was overfitted because of the validation loss sudden increases but the model seems to work fine.



An improvement for this project would be to collect more data and train the network on them and maybe creating a model with more encoders and decoders with increased depth and decreasing the learning rate.